Answering "If-What" Questions to Manage Batch Systems

Sainath Batthala*, Neminath Hubballi
Indian Institute of Technology, Indore.
Email: {cse1200106, neminath}@iiti.ac.in
*Student Author

Maitreya Natu, Vaishali Sadaphal
Tata Research Development and Design Centre,
Pune, India.
Email: {maitreya.natu, vaishali.sadaphal}@tcs.com

Abstract—**Batch systems are the backbone of all banking and financial industries. These systems are usually large and complex and often lack end-to-end transparency. As a result, any improvement in these systems presents several challenges. In this paper, we have addressed the problem of "if-what analysis" of batch systems. This analysis assists the system administrators to provide recommendations in order to improve batch performance, stability, and cost. We define the problem space, discuss solution ideas, and present initial results through a real-world case-study.**

I. Introduction

Batch systems play a significant role in today's business. Batch jobs perform end-of-day back-office tasks such as data integration, compliance check, analytics and reporting. A batch includes a set of jobs with precedence relations. Thus, a job can start only after all its defined predecessor jobs have completed execution. The business requirements enforce constraints on the latest time by which all jobs within the batch must complete. This time is referred to as the batch completion time. For instance, in an investment bank, the batch is usually required to complete before the trading market open time on the next business day. Delays and failures to meet the batch completion time result in heavy financial penalties and business outages. Service Level Agreements (SLAs) are defined for the business-critical batch jobs and business processes that define the batch completion time.

Batch systems observe delays or SLA violations due to various reasons such as increased workload, under-provisioned compute capacity, among others. Hence, the system administrators often require to apply various levers to reduce the batch completion time or to minimize the SLA violations of business critical jobs. Various levers can be applied to meet these objectives such as decreasing the run-time of jobs by adding more compute capacity, or changing the schedules of jobs to start early, among others. Each of these levers are associated with a cost and a benefit. Given the large number of jobs and dependencies, it is a non-trivial task to identify the right levers on the right jobs to meet the desired objective.

We refer to this analysis as "if-what analysis". This analysis answers the questions of the nature - "if I want to achieve an objective, then what actions should be taken?". The "if" part of question is generally dictated by objectives such as reduction in batch completion time, cost, SLA violations, etc. The answer for the "what" part of the question can span all aspects of batch environment such as infrastructure, batch schedule, job run-times, job dependencies, business processes, etc. Some examples of if-what questions are:

- If the run-time of a batch is to be reduced by 20% with the additional annual cost of less than USD 100,000, then what actions should be taken?
- If start-time and end-time of a batch is to be maintained within the desired SLAs in the event of doubling of workload, then what actions should be taken?

The ability to do if-what analysis improve the transparency of the batch systems and makes the batch environment agile and resilient. However, large number of batch jobs, complex dependencies, multiple combinations of levers to achieve objectives make if-what analysis quite challenging.

Past research [2] has focused on what-if analysis that evaluates the effect of various hypothetical scenarios of batch system changes such as effect of reducing run-time of a job, adding/removing jobs or dependencies, etc. Though what-if analysis is a powerful and important tool, it only answers a part of the big puzzle. What-if analysis evaluates the impact of introducing some change in system properties. On the other hand, if-what analysis is all about identifying the change that should be introduced to achieve a specific objective. In other words, if-what requires intelligent navigation through various what-if scenarios, asking the right what-if questions, and evaluating the costs and benefits of various levers to derive the optimal set of levers.

In this paper, we discuss the problem of if-what analysis. We define the solution space and identify different building blocks required to develop this solution. We present initial ideas on developing if-what solutions and demonstrate their effectiveness through a real-world case-study.

II. Design Rationale

In this section, we present initial ideas of the solution space. Figure 1 explains the data sources and the key building blocks of the proposed solution.

A. *Data sources:*

The solution uses feeds from following data sources:

1) System dependencies: We capture all the system dependencies viz. job-job, business process-job, and job-host dependencies. Batch jobs depend on each

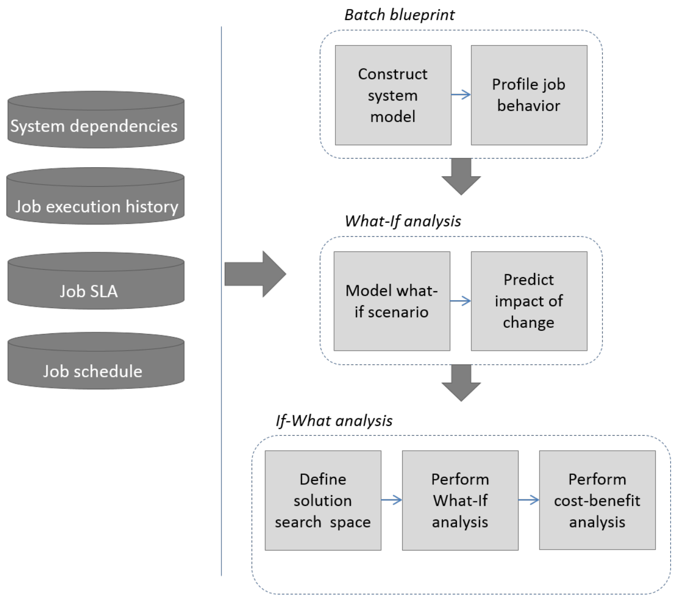

Fig. 1. Building blocks of the proposed solution.

other through precedence relationships. A job can start only after all its predecessor jobs have finished. Jobs run on predefined hosts. Jobs are grouped into business processes.

2) Job execution history: We analyze the past history of job executions. It captures the start-time, end-time, and run-time of each job instance executed in the past.
3) Job schedule: Each job is associated with an execution schedule. This schedule defines the condition for job execution with respect to day of week, day of month, day of year etc. Some jobs are defined to have conditions at start-time. For instance, all root jobs are defined to start on a fixed start-time. In some cases, business logic demands some intermediate nodes to have fixed start-times as well.
4) Many business-critical jobs are assigned service-level agreements which represent the business constraints on the start-time, end-time of job. Other commonly defined SLAs are MAXRUNALARM and MINRUNALARM that define the maximum time and minimum time a batch job execution should take.

B. Batch blueprint:

The first step in batch analytics is to construct a blueprint of the batch system. This involves modelling the batch system as a directed acyclic graph and annotate every node with attributes such as run-time, SLA thresholds, associated business process name, host name, etc. We construct the batch blueprint in two steps.

1) Construct system model: We model the batch system as a directed acyclic graph where nodes represent the batch jobs and edges represent the precedence relationships. We further enrich each job node with static attributes such as business-criticality, job schedule, job SLA (if any), associated business process, host name, etc. In addition, each job node also contains time-series attributes to capture historical data of workload, run-time, alerts, etc.

2) Profile job behavior: We next enrich the blueprint by modelling the normal behavior of each job. This normal behavior captures various attributes of a job such as trends and patterns in workload and run-time, change points, outliers, etc.

C. What-if analysis

Given the system model, we perform what-if analysis using two main building blocks.

1) Model what-if scenario: We first model a what-if scenario by changing the graph model. These changes could be in nodes, edges, or attributes of the graph model. We model the following what-if scenarios:

1) Addition or deletion job is modelled by adding or removing nodes and associated edges.
2) Change in run-time is modelled by changing the run-time attribute of one or more jobs.
3) Change in workload is modelled by computing the effect of change in workload on job run-time and modelling a change in job run-time.
4) Change in CPU is modelled by computing the effect of change in CPU on job run-time and modelling a change in job run-time.

2) Estimating effect of change: The impact of change is then computed by traversing the changed system model and re-annotating the start-time, run-time, and end-time of batch jobs. After recomputing the job execution parameters, these values are compared with SLAs to identify and annotate the jobs with SLA violations. We use topological or recursive traversal approach to find the impact.

D. If-what analysis

The solution to an if-what problem requires following capabilities:
(a) Identifying possible transformation levers
(b) Identifying potential solutions by combining these levers
(c) Estimating the effect of solutions by performing what-if analysis
(d) Searching for the best solutions to achieve the desired objective.

1) Define solution search space: This component identifies the potential set of jobs that can assist in achieving the desired objective of if-what analysis. A brute-force approach is to define the search space to explore all combinations of jobs to find the optimal solution. However, given the large search space, the brute-force approach is computationally impractical. Hence, we apply various heuristics to prune the search space.

2) Perform what-if analysis: This component runs what-if analysis for a given potential solution and estimates its impact on the batch system. It thus computes the start-time, end-time, and run-time of each batch job.

3) Perform cost-benefit analysis: Applying a what-if scenario is associated with a cost. For instance, decreasing run-time of a job might involve adding extra computing capacity. Similarly, a what-if scenario is associated with benefits such as decrease in batch run-time, decrease in SLA violations. This component evaluates various solution levers on various aspects of cost and benefit to identify the best solution.

III. Proposed Approach

As mentioned earlier in Section I, if-what analysis requires intelligent navigation through various what-if scenarios. Consider the following if-what question - "If the run-time of a batch is to be reduced by 20% with the additional annual cost less than USD 100,000, then what actions should be taken?". The solution space for this problem consists of many entities, such as jobs, dependencies, CPUs, etc. Each entity is associated with different transformation levers, such as increase/decrease run-time of jobs, add/delete dependencies, or increase/decrease CPUs.

The brute-force approach is to explore all possible combinations of levers on all jobs and estimate their impact. However, given the large scale of batch jobs (in the order of 100,000 jobs), this approach is not scalable. Hence, we propose various heuristics to intelligently prune irrelevant solutions and efficiently navigate through the relevant solution space.

We explain the proposed approach for if-what analysis using two transformation levers viz. reduce run-time of job, change start-times of root jobs. For clarity, we limit the below discussion to the if-what objective of finding best solutions to decrease the end-to-end batch completion time by Y units.

A. Reduce run-time of jobs

One of the most common levers to improve batch completion time is to reduce the run-time of one or more jobs. However, the challenge is to identify the smallest set of jobs whose reduction in run-time can lead to the desired impact on the batch completion time.

The if-what question that we consider is:

If the batch duration is to be reduced by Y units of time, then what is minimum number of jobs whose run-time needs to be reduced by X%?

A brute-force approach is to explore all combinations of batch jobs. However, this is not a scalable solution, hence we present a heuristic-based approach to intelligently prune the space and answer this if-what question.

1) Consider that *BatchEndTime* represents the current batch end-time, and *DesiredBatchEndTime* refers to *BatchEndTime - Y*, where Y is the amount of time by which the batch duration needs to be reduced.
2) Instead of analyzing all nodes, we first identify the leaf jobs that end after *DesiredBatchEndTime*. To meet the objective of reducing batch duration by Y units, it is imperative to ensure that these leaf jobs finish before the *DesiredfBatchEndTime*. We refer to these leaf jobs as *TargetLeafJobs*.
3) End time of *TargetLeafJobs* can be improved only by any changes in their upstream jobs. Thus, the search space is now reduced to *TargetLeafJobs* and their upstream jobs. We refer to this set as *UpstreamToTargetLeafJobs*.

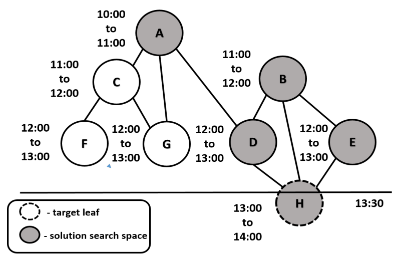

Fig. 2. Example to demonstrate the effect of reduction in run-time of jobs.

4) We now systematically apply apriori search to explore combinations of the jobs in the set *UpstreamToTargetLeafJobs*. We start with sets of size 1 and incrementally increase the combination size to 2, 3, and so on.
5) For each combination C, we apply what-if analysis to compute the impact of reducing run-time of jobs in C by X% on the overall batch duration.
6) The combination that provides the desired objective of reducing batch duration by Y units is identified as the solution for the if-what question.

1) Example: We next illustrate the above method with an example. Consider the batch system shown in Figure 2. The if-what objective is to reduce batch duration by 30 minutes by reducing the run-time of minimum number of jobs by 50%.

1) *BatchEndTime* = 14:00 and *DesiredBatchEndTime* = 13:30.
2) The only target leaf in this case is job H that ends after *DesiredBatchEndTime*.
3) Thus, the pruned search space now limits to H and its upstream jobs. Thus, *UpstreamToTargetLeafJobs* = $\{A, B, D, E, H\}$.
4) We first explore subsets of size 1 and perform what-if analysis to estimate their impact on batch duration. The impact of subsets {A}, {D}, {E}, {B}, {H} on batch duration are 0 mins, 0 mins, 0 mins, 30 mins, 30 mins, respectively.
5) It can be observed that the batch duration can be reduced by 30 minutes by reducing run-time of job B or H by 50%.

B. Change start-times of root jobs

We next present another scenario where we reduce the batch end-time by changing the start-time of the root jobs. In this section, we demonstrate an approach to identify the minimum number of root jobs that can lead to the desired impact on batch end-time.

The if-what question that we consider is:

If the batch end-time is to be reduced by Y units of time, then what is the minimum number of root jobs whose start-time needs to be reduced?

Below, we present the steps to identify these root jobs.

1) Similar to previous section, we define *BatchEndTime*, and *DesiredBatchEndTime*, and *TargetLeafJobs*.

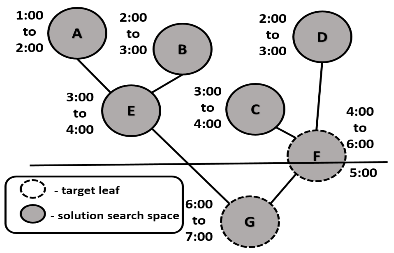

Fig. 3. Example to demonstrate the effect of reduction in start-time of root jobs.

2) Similar to previous section, we focus only on the *TargetLeafJobs* and their upstream jobs.
3) We iteratively select one of the leaf jobs in *TargetLeafJobs*. For each such leaf job we compute the amount of time by which the start-time of root jobs in its upstream tree needs to be reduced, such that the leaf job finishes before the *DesiredBatchEndTime*.
4) Given a leaf node L, the time by which the start-time of the root jobs in its upstream tree need to be reduced is computed as follows:
   - Consider a node N with parent P. In order to reduce the end-time of node N by Y units, the start-time of P needs to be reduced by Y units. However, often there exists a slack time S_{PN} between the end of job P and start of job N. Slack is the delay between the start of a child job after the end of the parent job. This slack can be removed. Hence, the start-time of P needs to be reduced by $Y - S_{PN}$ units.
   - In cases where a parent node P has more than one child nodes, then the desired reduction in start-time is computed as the maximum of the desired reduction in start-time required for each of its child nodes. ReductionInStartTime(P) is computed as follows:

     $$max(\forall_{N_i \in ChildNodes(P)}(Y_{N_i} - S_{PN_i})) \quad (1)$$

     where Y_{N_i} refers to the amount of time by which end-time of node N_i needs to be reduced, and S_{PN_i} refers to the slack time between nodes P and N_i.
   - Given a leaf job L, we perform reverse topological sort of its upstream jobs and compute the desired start-time of each of the upstream jobs using the above concept. We thus derive the desired start-time of the root-jobs in the upstream tree of the leaf job L.
5) We next perform what-if analysis to compute the impact of reducing start-time of the identified root jobs on the batch end-time.
6) If the desired objective is not met, then we repeat the above steps for the remaining set of *TargetLeafJobs*.

1) Example: We explain the above approach with an example.

Consider the batch system shown in Figure 3. The objective is to decrease batch end-time by 2 hours by changing the start-time of the minimum number of root jobs.

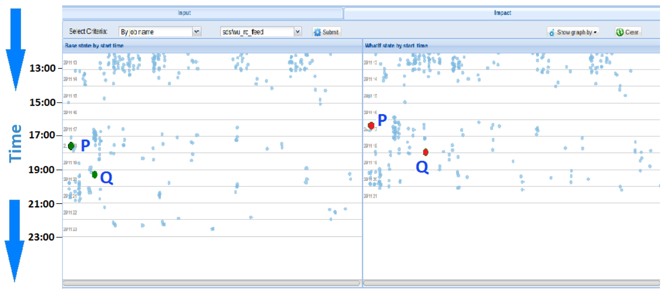

Fig. 4. Comparision of batch schedule before and after reducing the run-time of P and Q.

In this case, the target leaves are {F,G}. The search space is upstream of {F,G}, which is {A,B,C,D,E,F,G}.

1) *BatchEndTime* = 07:00 and *DesiredBatchEndTime* = 05:00.
2) The target leaf jobs that end after *DesiredBatchEndTime* are $\{F, G\}$.
3) For each leaf, we traverse the search space of its upstream jobs in reverse topological order.
4) We derive that batch end-time can be reduced by 2 hours by reducing the start-time of C by 2 hours and D by 1 hour.

IV. Experimental Results

We share the results for if-what analysis with 2 transformation levers on 2 real world batch systems of leading banks in the US. The job names have been masked for the purpose of confidentiality.

A. *Reduce run-time of job*

The batch consists of 270 jobs, 11 business processes, 444 precedence relationships. There are 7 root jobs and 59 leaf jobs in the batch. The batch ran for 9.74 hours.

We analyze the impact of reducing run-time of jobs. We compute the minimum number of jobs whose run-time should be reduced by 50%, so that batch duration is reduced by 1 hour. Initially, the batch is starting at 13:45:00 PM and ending at 23:29:41 PM. The aim is to make sure batch ends on or before 22:29:41 PM. Initially, the search space consists of all 270 jobs. However, only those leaf jobs need to finish earlier which are ending after 22:29:41 AM. As mentioned earlier in Section II-D, these leaf jobs are called target leaves. In this case, there are only 5 target leaves. As we need to move up only these 5 jobs, we prune the solution search space from all 270 jobs to upstream of target leaves which has only 36 jobs. Now, we identify minimum number of jobs from search space whose run-time reduction by 50% can meet our goal. We compute the impact of reducing run-time of all subsets of jobs of size 1 by performing what-if analysis. This did not achieve the objective. We then computed the impact of all subsets of size 2 and so on. Finally, the objective was achieved by changing the run-time of two jobs P, Q by 50% (P's run-time changed from 5334 to 2677 seconds and Q's run-time changed from 6482 to 3241 seconds). This can reduce the batch duration by 1.6 hours. Figure 4 shows the time series of batch before and after changing the run-time of jobs P and Q.

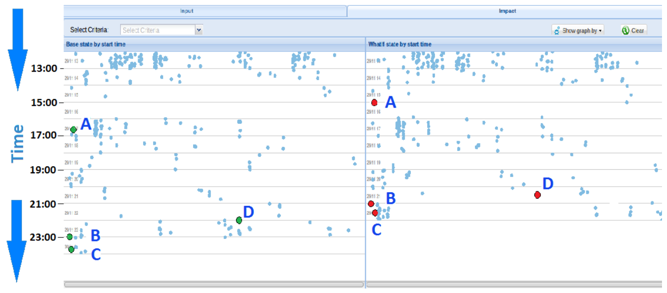

Fig. 5. Comparision of batch schedule before and after reducing the start-time of A, B, C and D.

Each blue circle represents a job in the batch. The jobs P and Q are highlighted with green (before) and red (after) colors. The reduction in batch duration by the upward movement of jobs after changing run-time of P and Q can be clearly observed.

B. *Change start-times of roots*

We consider another batch with 276 jobs, 13 business processes, 449 dependencies. The batch is running for 11.31 hours.

We reduce the batch end-time by starting the root jobs earlier. We compute minimum number of root jobs that should start earlier to finish the batch earier by 2 hours. There are 59 target leaves. We traverse the upstream of target leaves in reverse topological order and calculate the upward movement of nodes. We found that by starting root A earlier by by 4589 seconds, B by 5564 seconds, C by 7200 seconds, D by 3555 seconds, batch can finish earlier by 2 hours. Out of 8 roots only 4 roots had to start earlier than the current time.

Figure 5 shows the batch run before and after reducing the start-time of root jobs A, B, C and D. Each blue circle represents a job in the batch. The jobs $\{$A,B,C,D$\}$ are highlighted with green (before) and red (after) colors. The reduction in batch end-time by the upward movement of jobs after changing start-time of of A, B, C and D can be clearly observed.

V. Related Work

Work has been done in the past on what-if analysis for non-batch distributed systems. Authors in WISE [6] answer questions related to deployment and configuration for CDNs. [4] predicts the impact of workload change in complex cloud applications. [5] presents Predico, a workload-based what-if analysis system that uses commonly available monitoring information in large scale systems to ask a variety of workload-based "what-if" queries about the system.

Authors in [1] focus on scheduling of batch jobs on multi-processor systems such that batch finishes in minimum time with minimum resource utilization. [3] explains methods to perform if-what analysis on datacenters.

We leverage the past work and build on top of it to develop the capability of if-what analysis.

VI. Conclusion and future work

We presented the initial ideas to address if-what analysis of batch systems. We discussed an approach to model the operations of a batch system and answered if-what questions. We demonstrated the effectiveness of the proposed approach through experiments.

There are several open issues and unanswered problems in this space. As a part of our ongoing research, we will address a more comprehensive set of if-what questions and define a wider variety of solution levers. Our approach to if-what analysis is not limited to batch systems and we strongly believe that it is generic and can be extended to transactional systems as well.

Currently, we are performing if-what analysis on past batch runs. An interesting direction to pursue is to perform if-what analysis on predicted batch schedules and recommend proactive measures to avoid business penalties.

We believe that if-what analysis can provide a powerful lever to open several avenues for strategy planning for technology and business.

References

[1] Rushi Agrawal and Vaishali Sadaphal. Batch systems: Optimal scheduling and processor optimization. *Student Research Symposium, HiPC - IEEE International Conference on High Performance Computing*, 2011.

[2] Pushkar Gupta, Aruna Malapati, Maitreya Natu, and Vaishali Sadaphal. Toward predictable batch systems using what-if analysis. *Student Research Symposium, HiPC - IEEE International Conference on High Performance Computing*, 2014.

[3] Vaishali Sadaphal, Maitreya Natu, Harrick Vin, and Prashant Shenoy. If-what analysis for data center transformations. In *Proceedings of the Workshop on Posters and Demos Track*, PDT '11, pages 13:1–13:2, New York, NY, USA, 2011. ACM.

[4] Rahul Singh, Prashant Shenoy, Maitreya Natu, Vaishali Sadaphal, and Harrick Vin. Analytical modeling for what-if analysis in complex cloud computing applications. *SIGMETRICS Perform. Eval. Rev.*, 40(4):53–62, April 2013.

[5] Rahul Singh, Prashant J. Shenoy, Maitreya Natu, Vaishali P. Sadaphal, and Harrick M. Vin. Predico: A system for what-if analysis in complex data center applications. In *Middleware 2011 - ACM/IFIP/USENIX 12th International Middleware Conference, Lisbon, Portugal, December 12-16, 2011. Proceedings*, pages 123–142, 2011.

[6] Muhammad Mukarram Bin Tariq, Amgad Zeitoun, Vytautas Valancius, Nick Feamster, and Mostafa H. Ammar. Answering what-if deployment and configuration questions with wise. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008*, pages 99–110, 2008.